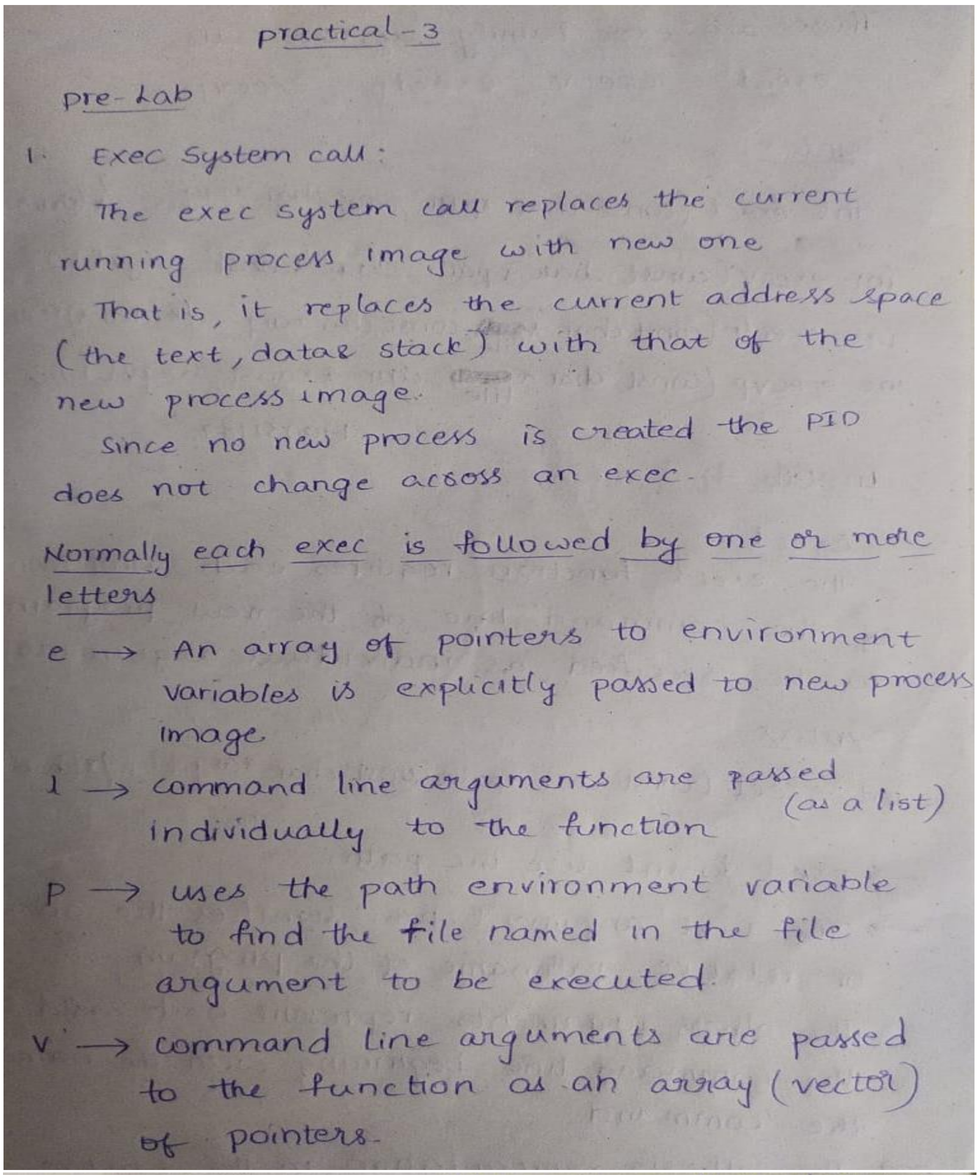
Lab Experiment – 3

Pre-Lab 3:

The exec family(**execl, execv, execlp, execvp**) of library functions replace the calling process's code, data, and stack from the executable whose pathname is stored in path



practical-3

pre-Lab

1. Exec System call:

The exec system call replaces the current running process image with new one

That is, it replaces the current address space (the text, data & stack) with that of the new process image.

Since no new process is created the PID does not change across an exec.

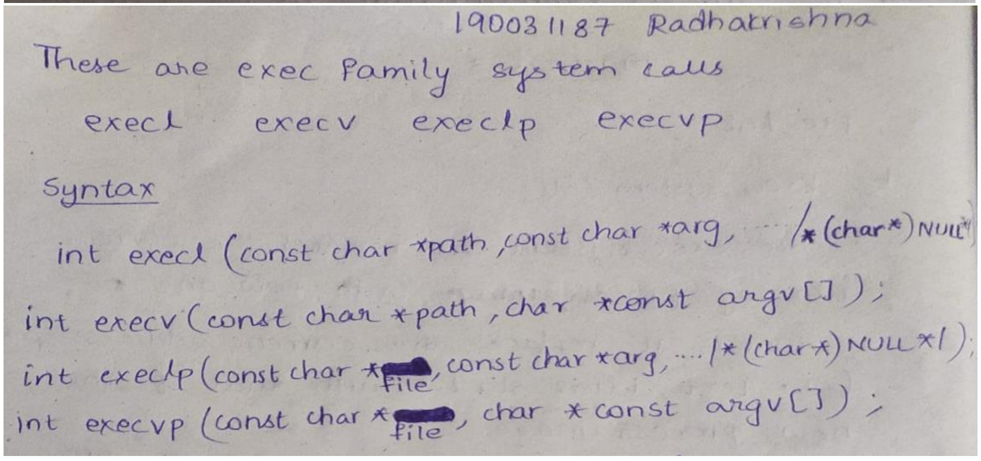
Normally each exec is followed by one or more letters

e → An array of pointers to environment variables is explicitly passed to new process image

l → command line arguments are passed individually to the function (as a list)

p → uses the path environment variable to find the file named in the file argument to be executed.

v → command line arguments are passed to the function as an array (vector) of pointers.



190031187 Radhakrishna

These are exec Family system calls

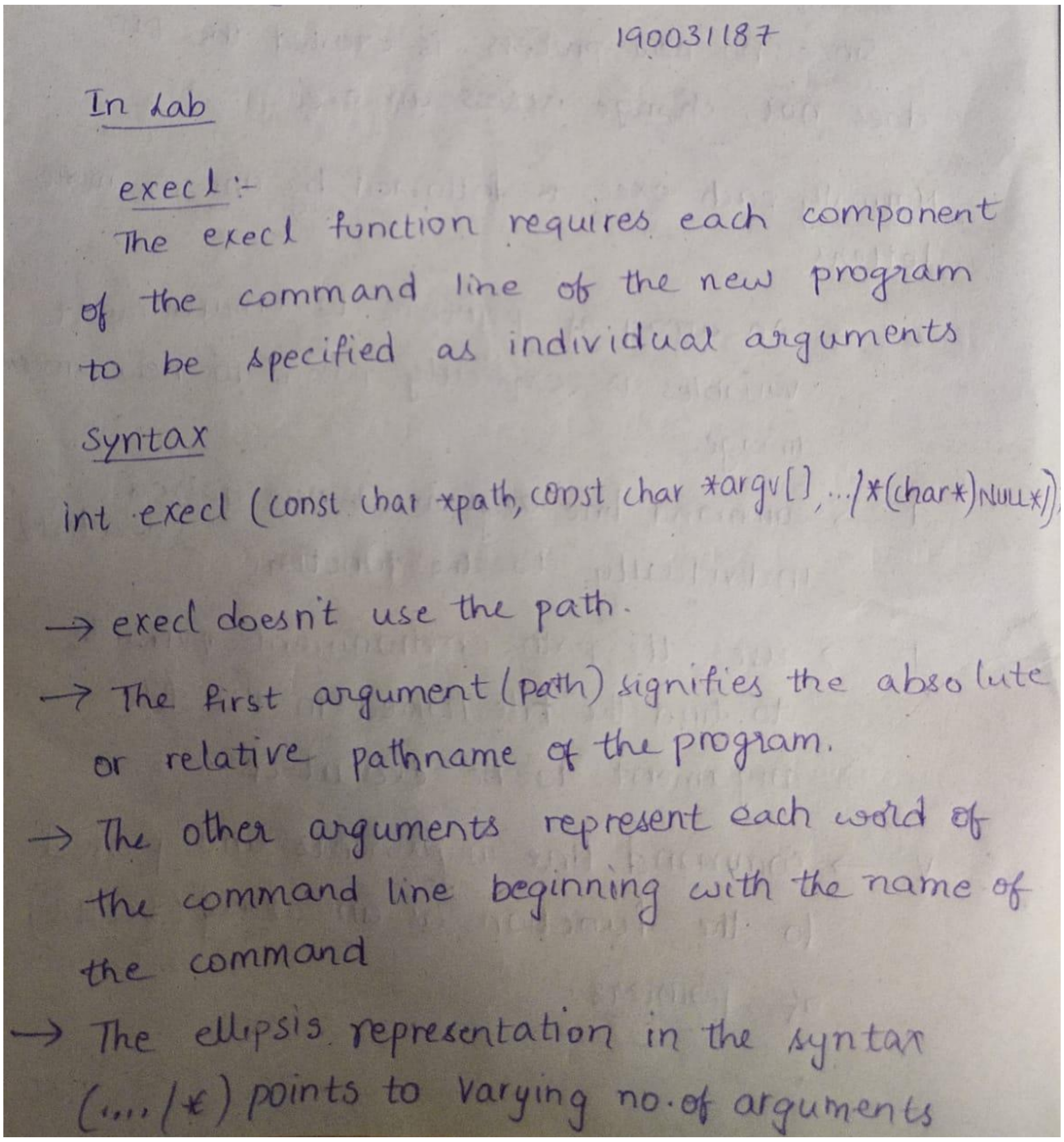
execl execv execlp execvp

Syntax

```
int execl (const char *path, const char *arg, ... /* (char*) NULL*/
int execv (const char *path, char *const argv[]);
int execlp (const char *file, const char *arg, .... /* (char*) NULL */);
int execvp (const char *file, char * const argv[]);
```

In-Lab 3:

- execl *and* execv, Gathering the exit Status using wait with standard input and output redirection.



190031187

In Lab

execl :-

The execl function requires each component of the command line of the new program to be specified as individual arguments

Syntax

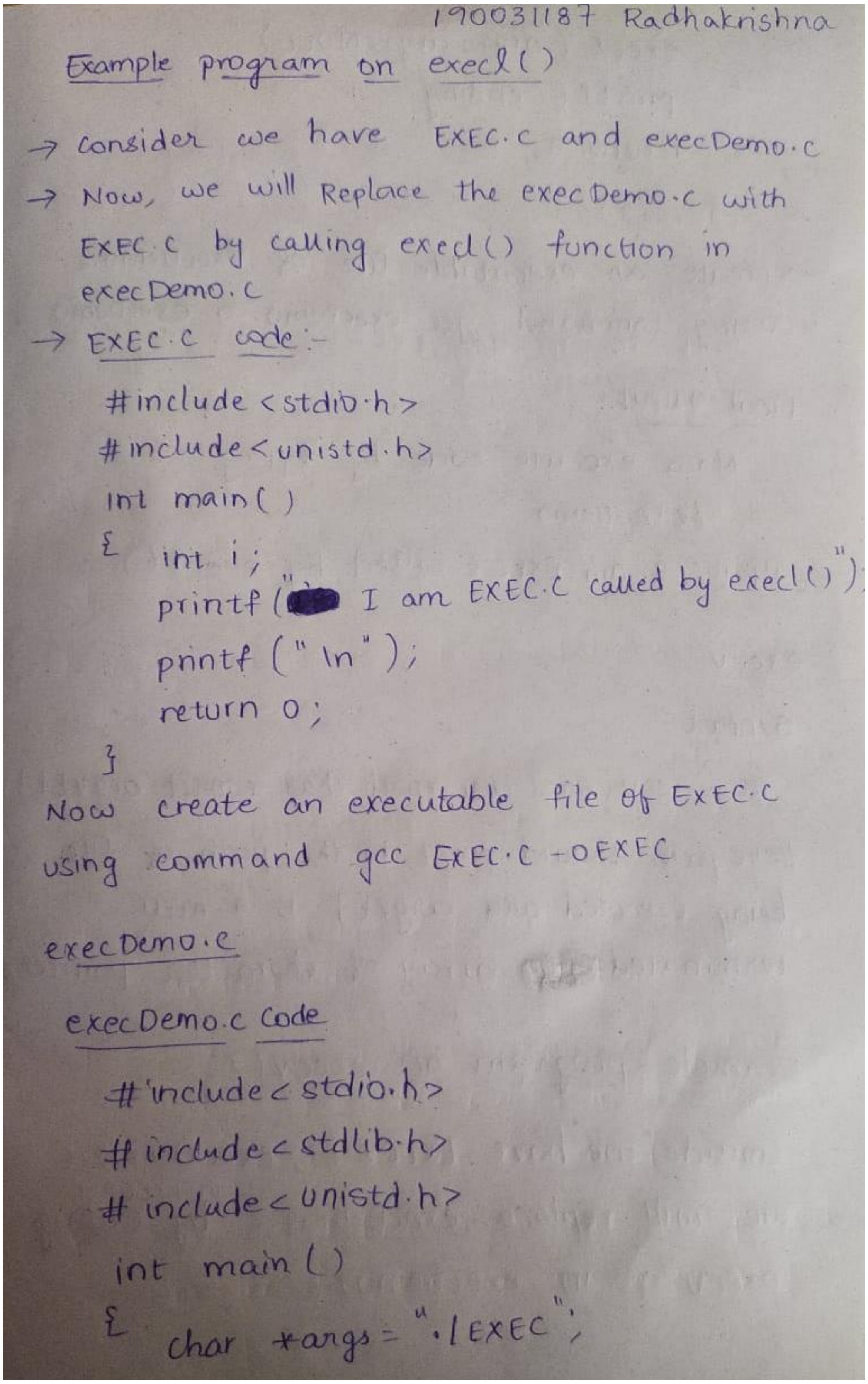
int execl (const char *path, const char *argv[], .../*(char*)NULL*/);

→ execl doesn't use the path.

→ The first argument (path) signifies the absolute or relative pathname of the program.

→ The other arguments represent each word of the command line beginning with the name of the command

→ The ellipsis representation in the syntax (..., /*) points to varying no. of arguments



190031187 Radhakrishna

Example program on execl()

→ consider we have EXEC.c and execDemo.c

→ Now, we will Replace the execDemo.c with EXEC.c by calling execl() function in execDemo.c

→ EXEC.c code:-

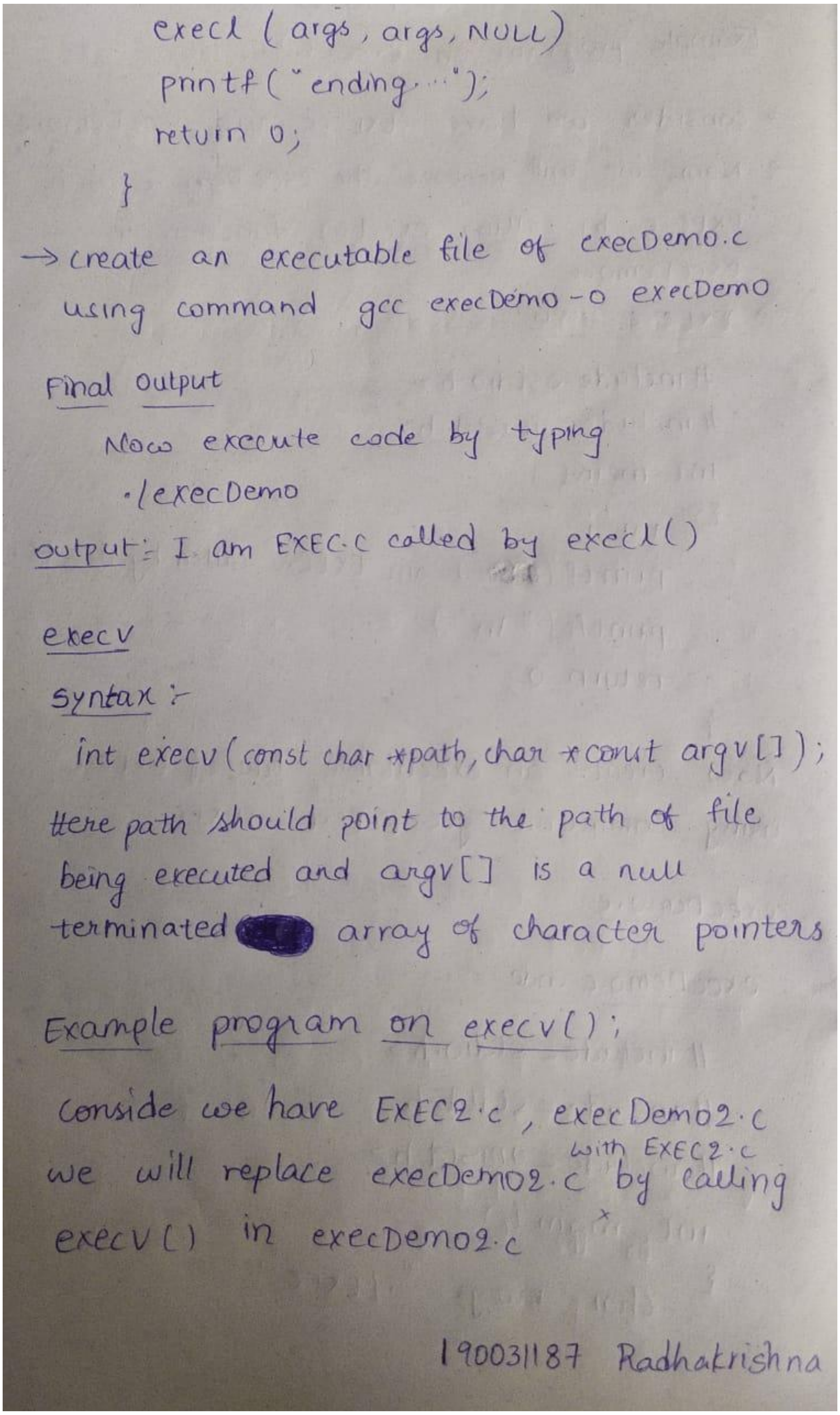
```
#include <stdio.h>
#include<unistd.h>
int main( )
{  int i;
   printf(" I am EXEC.C called by execl()");
   printf (" \n");
   return 0;
}
```

Now create an executable file of EXEC.c using command gcc EXEC.c -oEXEC

execDemo.c

execDemo.c Code

```
#include< stdio.h>
#include< stdlib.h>
#include< unistd.h>
int main ( )
{  char *args= "./EXEC";
```



```
    execl ( args , args, NULL)
    printf( "ending...");
    retuin 0;
}
```

→ create an executable file of execDemo.c using command gcc execDemo -o execDemo

Final Output

Now execute code by typing

./execDemo

output:- I am EXEC.c called by execl()

execV

syntax :-

int execv(const char *path, char *const argv[]);

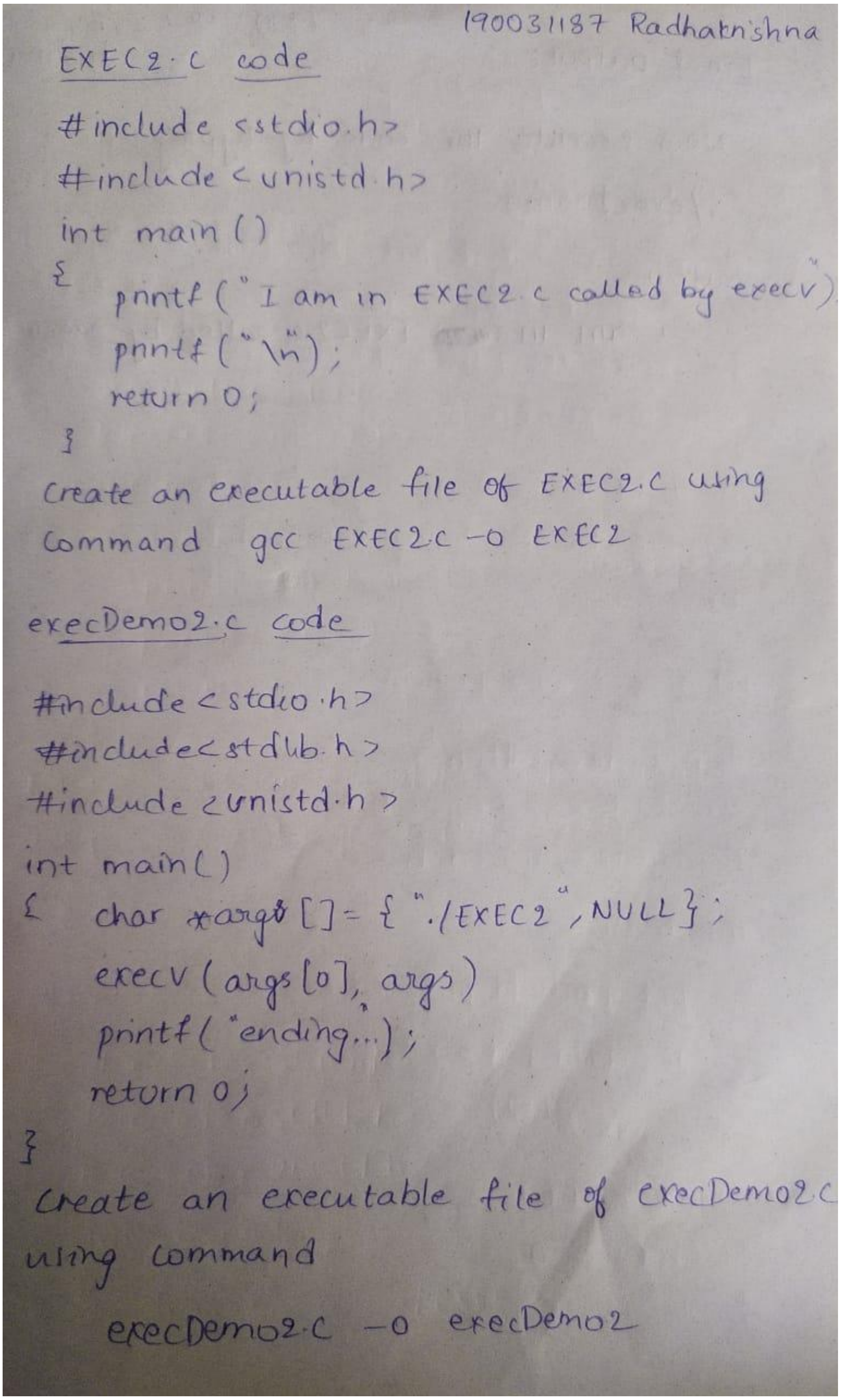
Here path should point to the path of file being executed and argv[] is a null terminated array of character pointers

Example program on execv();

conside we have EXEC2.c , execDemo2.c

we will replace execDemo2.c with EXEC2.c by calling execV() in execDemo2.c

190031187 Radhakrishna



190031187 Radhakrishna

EXEC2.C code

```
#include <stdio.h>
#include <unistd.h>
int main ()
{
    printf ("I am in EXEC2.c called by execv")
    printf ("\n");
    return 0;
}
```

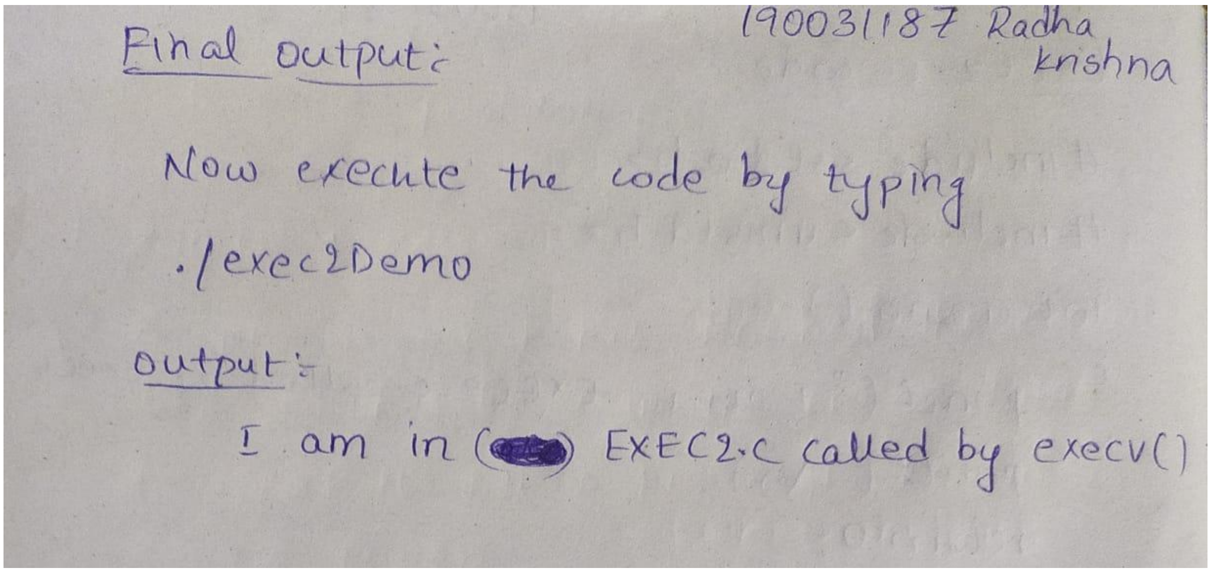
Create an executable file of EXEC2.C using
Command gcc EXEC2.C -o EXEC2

execDemo2.c code

```
#include <stdio.h>
#include<stdlib.h>
#include <unistd.h>
int main()
{   char ~~*args~~ [] = { "./EXEC2", NULL };
    execv (args[0], args)
    printf ("ending...);
    return 0;
}
```

Create an executable file of execDemo2C
using command

execDemo2.C -o execDemo2



190031187 Radha krishna

Final output:

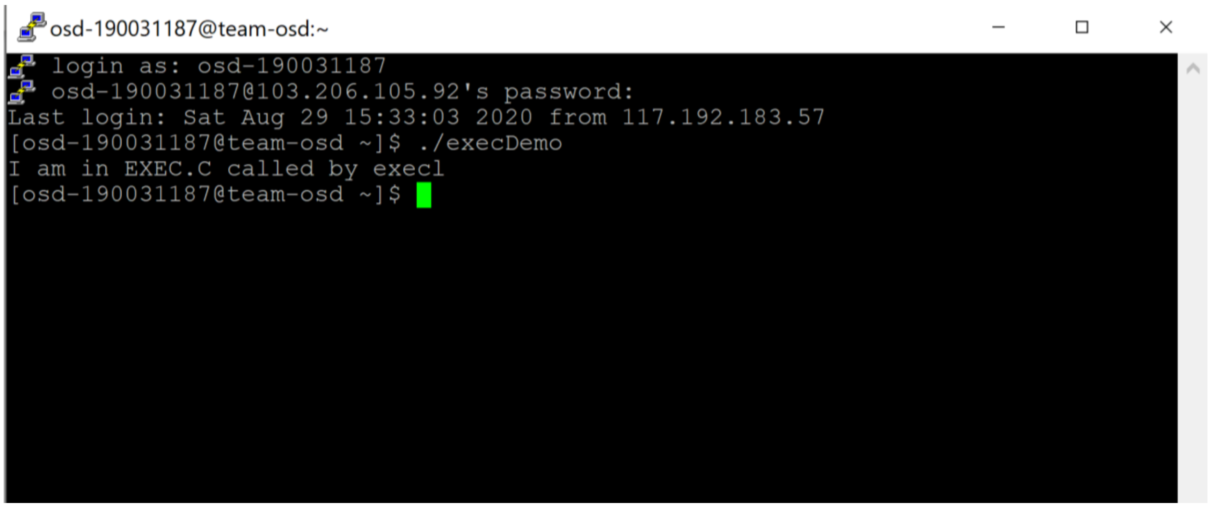
Now execute the code by typing

./exec2Demo

output:

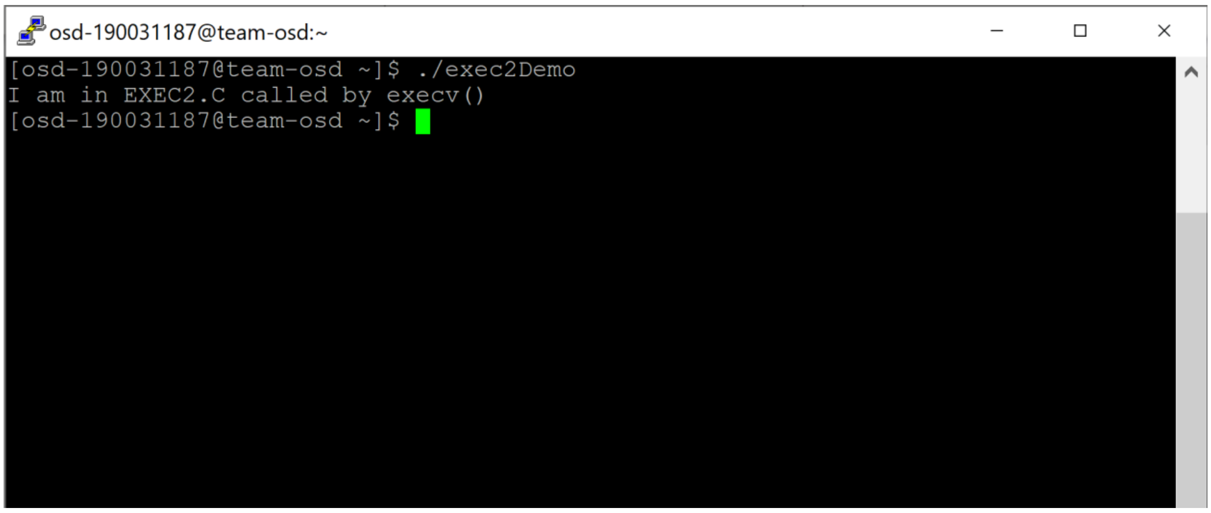
I am in ([illegible]) EXEC2.C called by execv()

execl() OUTPUT



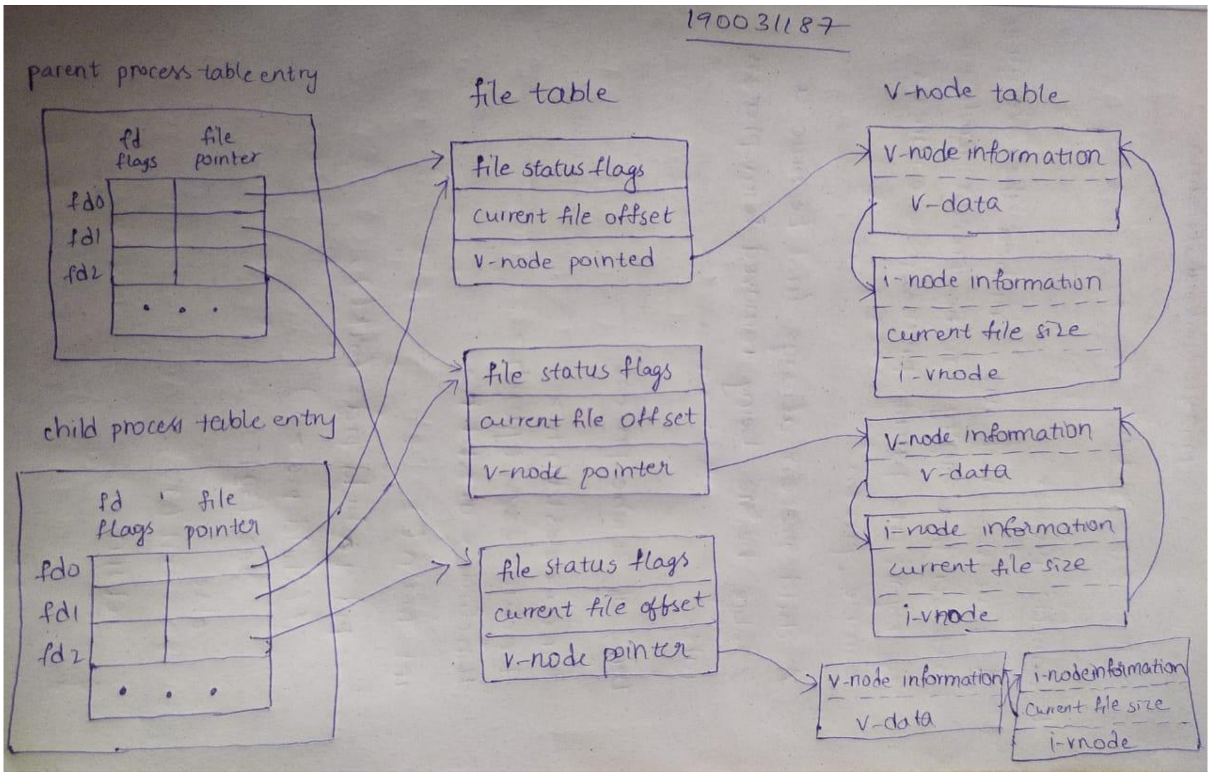
```
osd-190031187@team-osd:~
login as: osd-190031187
osd-190031187@103.206.105.92's password:
Last login: Sat Aug 29 15:33:03 2020 from 117.192.183.57
[osd-190031187@team-osd ~]$ ./execDemo
I am in EXEC.C called by execl
[osd-190031187@team-osd ~]$
```

execv() OUTPUT



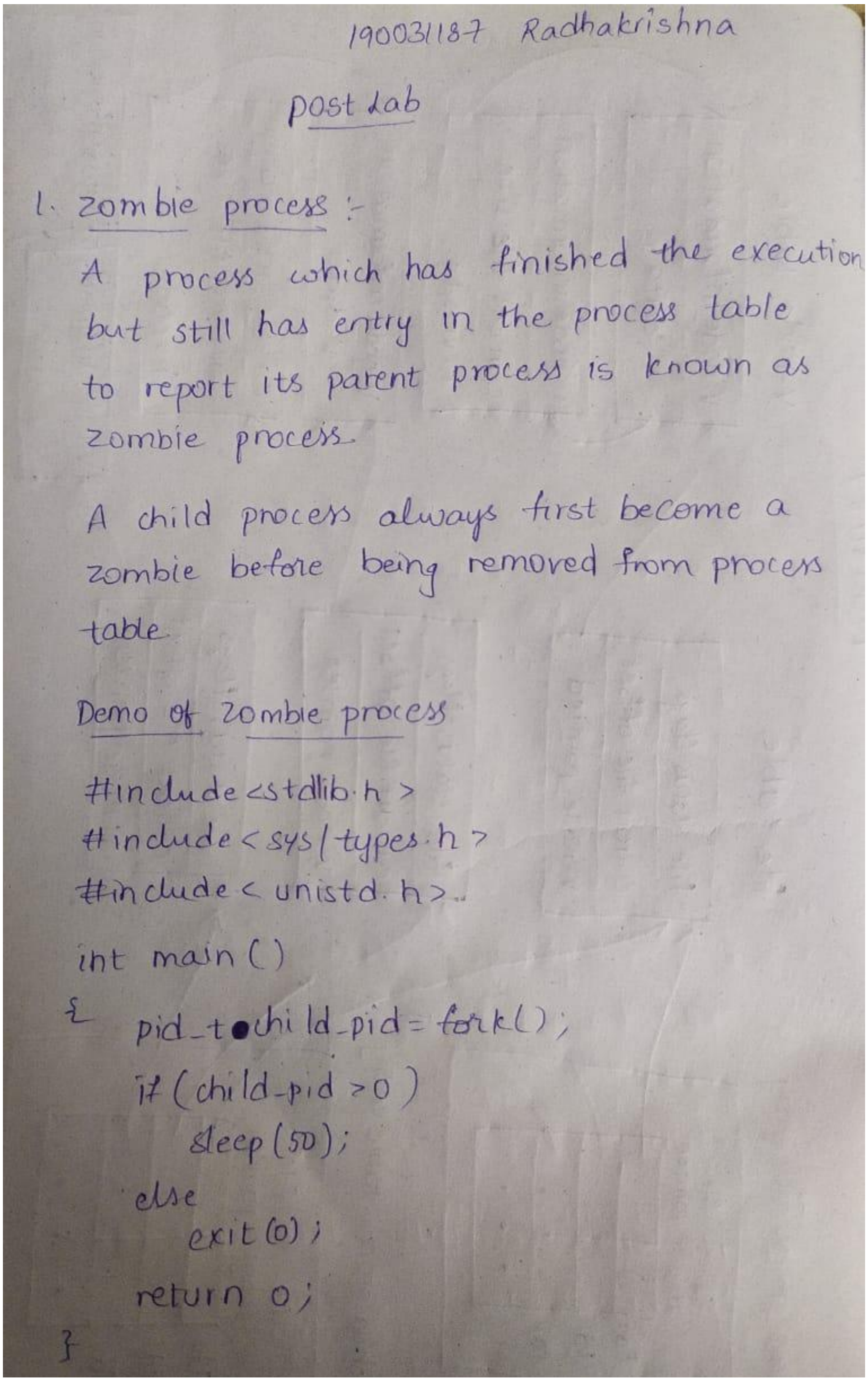
```
osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ ./exec2Demo
I am in EXEC2.C called by execv()
[osd-190031187@team-osd ~]$
```

- Show a pictorial arrangement - Sharing of open files between parent and child after fork



Post-Lab 3:

- orphan.c, zombie.c: create orphan and processes



190031187 Radhakrishna

post Lab

1. zombie process :-

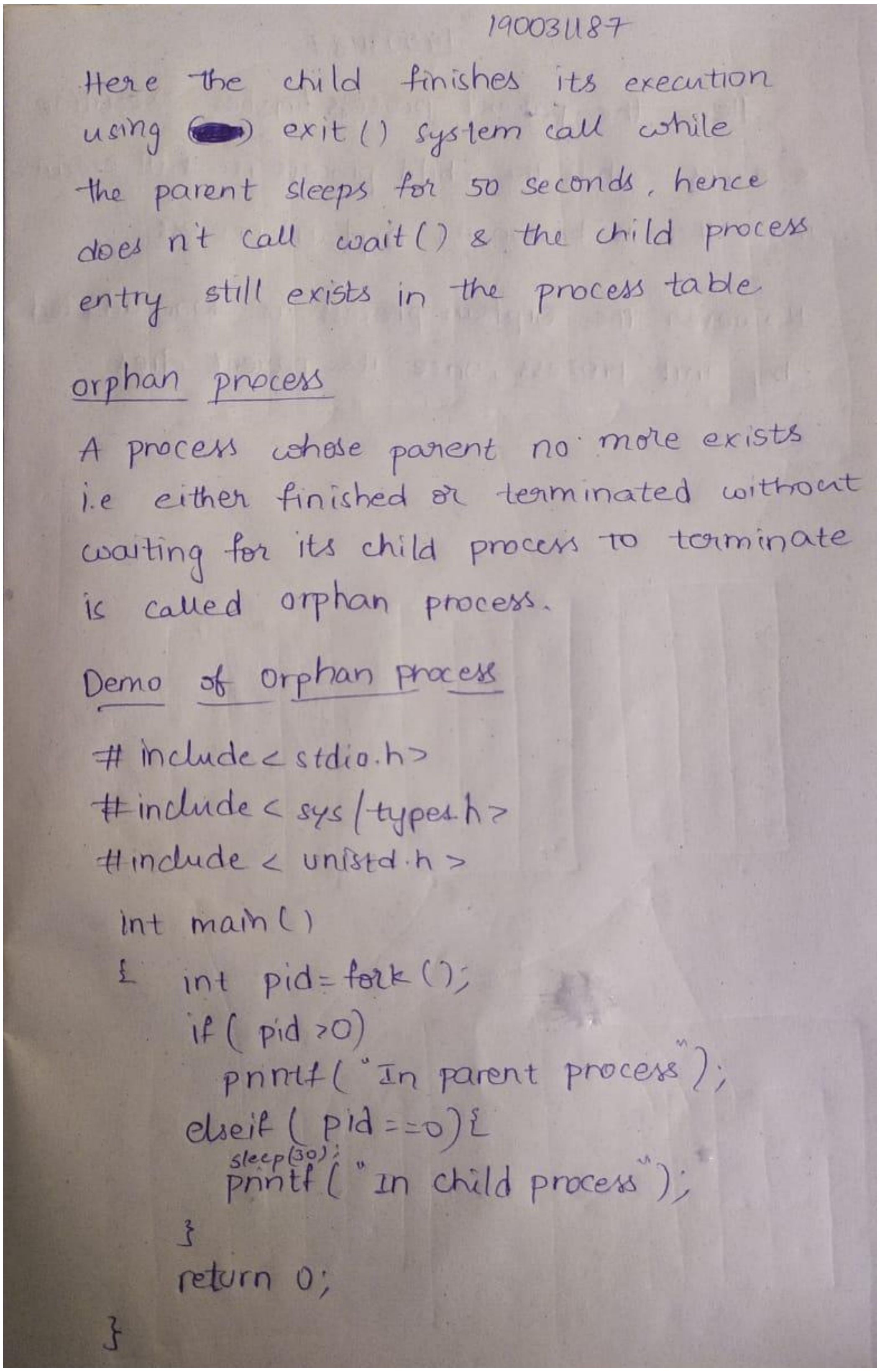
A process which has finished the execution but still has entry in the process table to report its parent process is known as zombie process.

A child process always first become a zombie before being removed from process table

Demo of zombie process

```
#include<stdlib.h >
#include<sys/types.h >
#include< unistd.h>..

int main()
{  pid_t child_pid= fork();
   if (child_pid >0 )
      sleep(50);
   else
      exit(0);
   return 0;
}
```



190031187

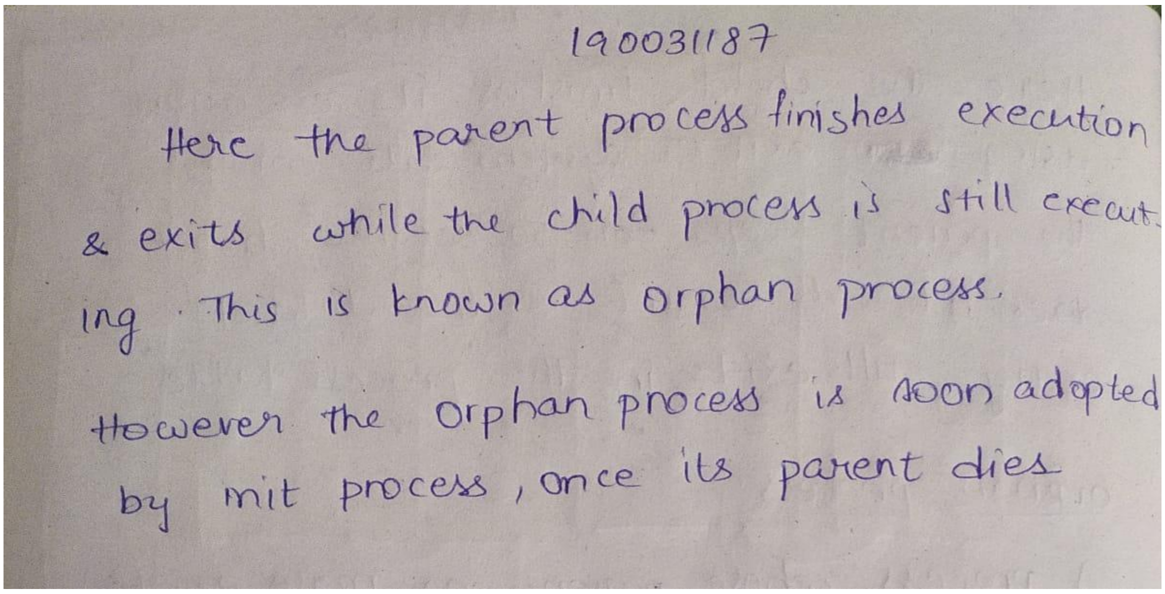
Here the child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence does n't call wait() & the child process entry still exists in the process table

orphan process

A process whose parent no more exists i.e either finished or terminated without waiting for its child process to terminate is called orphan process.

Demo of orphan process

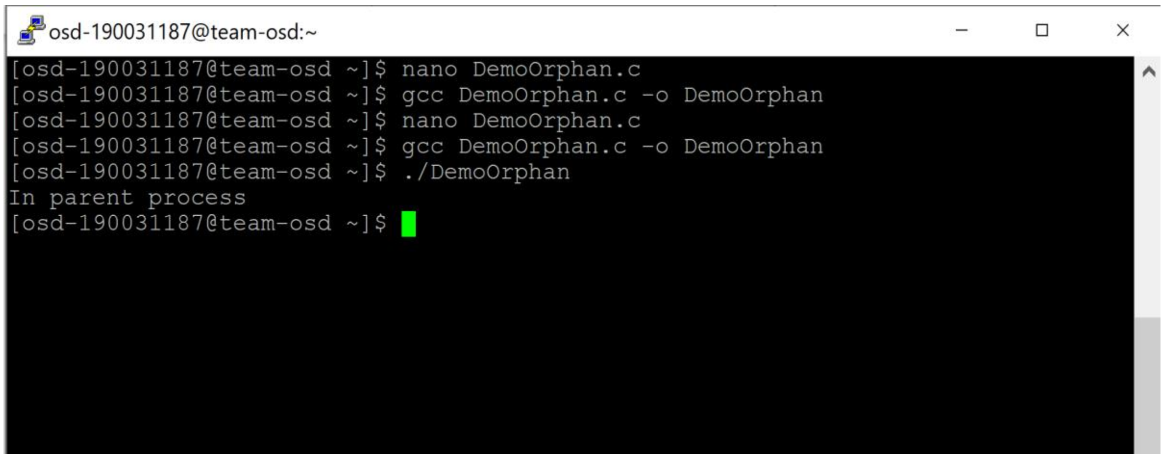
```
# include< stdio.h>
#include < sys/types.h>
#include < unistd.h >
 int main()
 { int pid= fork();
   if( pid >0)
     printf("In parent process");
   elseif ( pid ==0){
     sleep(30);
     printf( "In child process");
   }
   return 0;
 }
```


190031187

Here the parent process finishes execution & exits while the child process is still executing. This is known as orphan process.

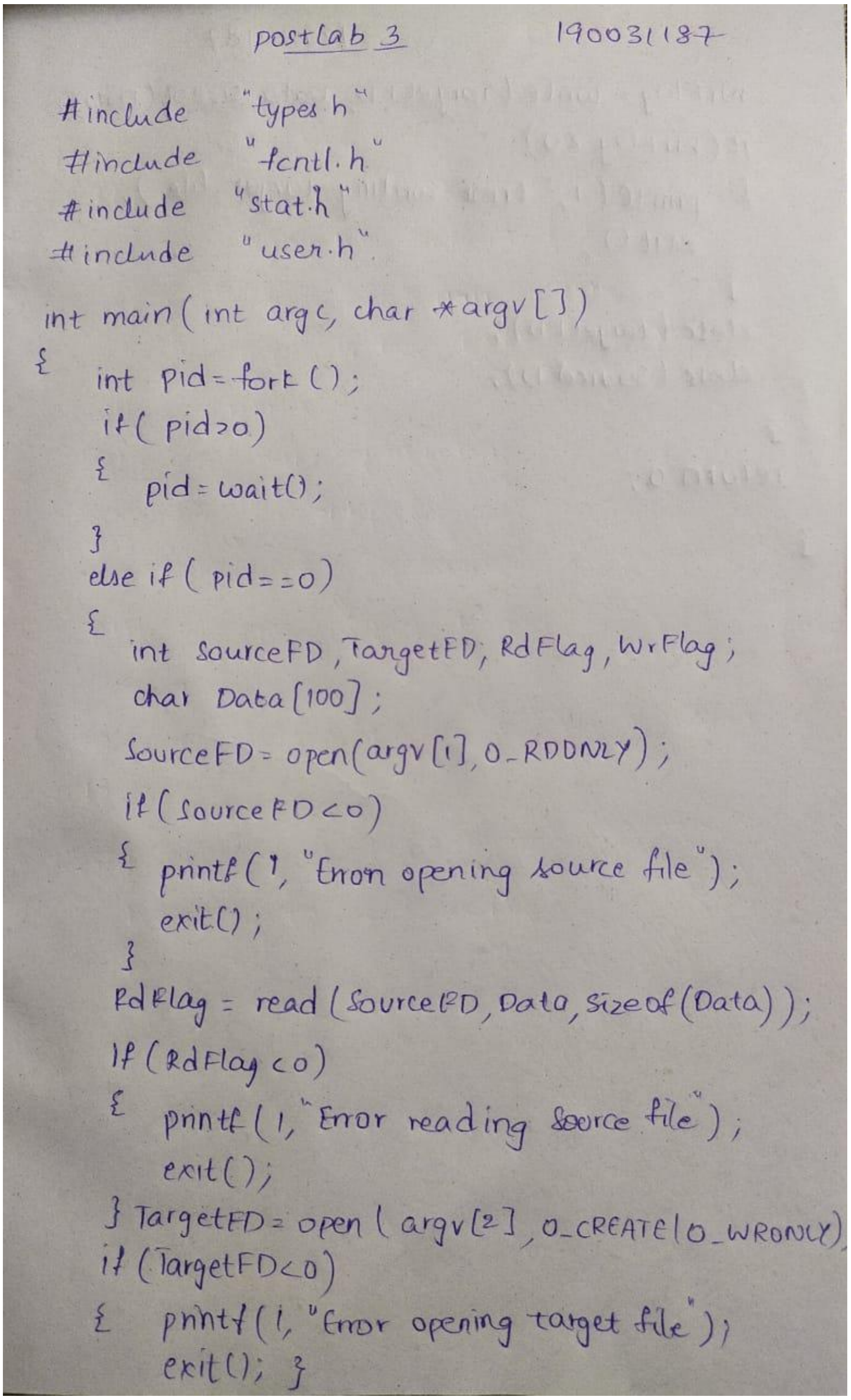
However the orphan process is soon adopted by init process, once its parent dies

DEMO OF ORPHAN PROCESS OUTPUT


osd-190031187@team-osd:~

```
[osd-190031187@team-osd ~]$ nano DemoOrphan.c
[osd-190031187@team-osd ~]$ gcc DemoOrphan.c -o DemoOrphan
[osd-190031187@team-osd ~]$ nano DemoOrphan.c
[osd-190031187@team-osd ~]$ gcc DemoOrphan.c -o DemoOrphan
[osd-190031187@team-osd ~]$ ./DemoOrphan
In parent process
[osd-190031187@team-osd ~]$
```

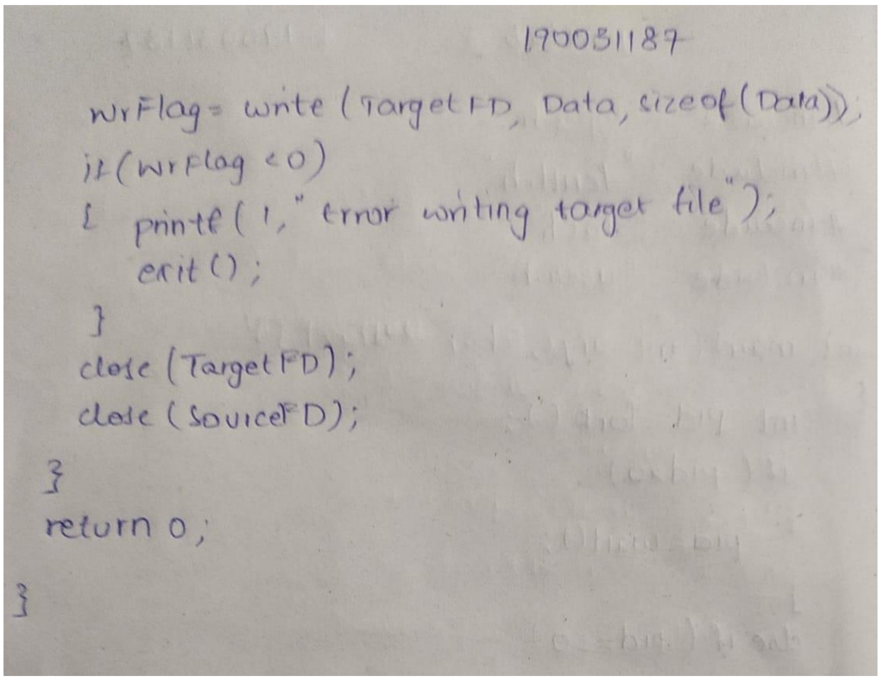
- Program that creates a new Process to Copy File



postlab 3 190031187

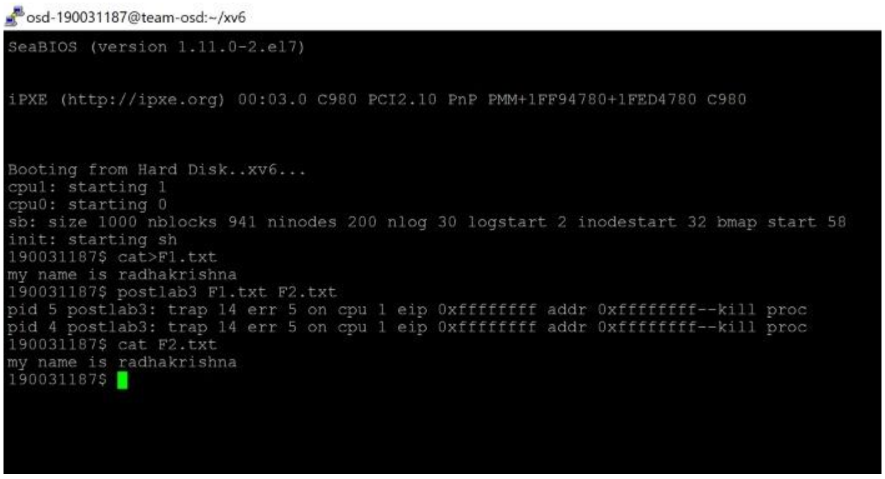
```
#include "types.h"
#include "fcntl.h"
#include "stat.h"
#include "user.h"

int main(int argc, char *argv[])
{   int pid=fork();
    if(pid>0)
    {   pid=wait();
    }
    else if(pid==0)
    {
      int SourceFD,TargetFD,RdFlag,WrFlag;
      char Data[100];
      SourceFD=open(argv[1],O_RDONLY);
      if(SourceFD<0)
      { printf(1,"Error opening source file");
        exit();
      }
      RdFlag = read(SourceFD,Data,sizeof(Data));
      If(RdFlag<0)
      { printf(1,"Error reading source file");
        exit();
      } TargetFD=open(argv[2],O_CREATE|O_WRONLY)
    if(TargetFD<0)
    { printf(1,"Error opening target file");
      exit(); }
```



190031187

```
wrFlag= write (TargetFD, Data, sizeof(Data));
if(wrFlag <0)
{  printf( 1," Error writing target file");
   exit();
}
close (TargetFD);
close (SourceFD);
}
return 0;
}
```



osd-190031187@team-osd:~/xv6

```
SeaBIOS (version 1.11.0-2.el7)


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF94780+1FED4780 C980



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
190031187$ cat>F1.txt
my name is radhakrishna
190031187$ postlab3 F1.txt F2.txt
pid 5 postlab3: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
pid 4 postlab3: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
190031187$ cat F2.txt
my name is radhakrishna
190031187$
```